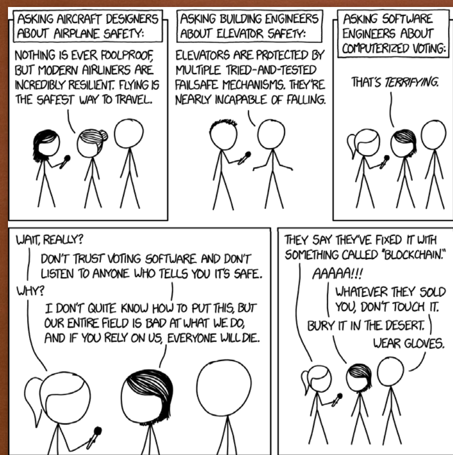
MOVING ON TO SOFTWARE

WITH A BIT OF NON VON NEUMANN



https://xkcd.com/2030/

LEARNING OBJECTIVES

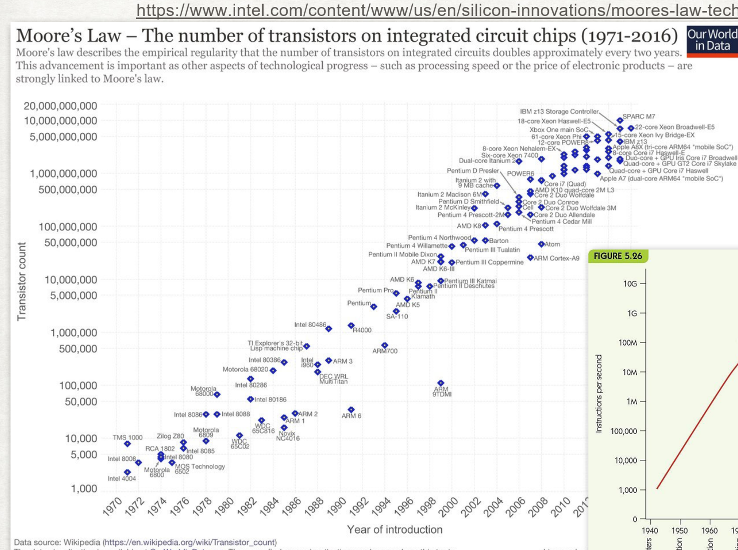
- Non Von Neumann architectures and why we need them
- Compare the virtual machine created for the user by system software with the naked machine
- Describe the different types of system software
- Explain the benefits of writing programs in assembly language rather than machine language

NON–VON NEUMANN ARCHITECTURES (1 OF 6)

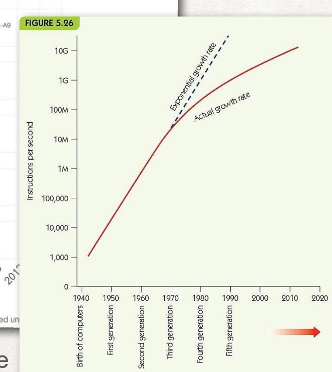
- Problems to solve are always larger
- Computer chip speeds no longer increase exponentially
- Reducing size puts gates closer together, faster
  - Speed of light pertains to signals through wire
  - Cannot put gates much closer together
  - Heat production increases too fast
- **Von Neumann bottleneck**: inability of sequential machines to handle larger problems

NON–VON NEUMANN ARCHITECTURES (2 OF 6)

https://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html



Transistors



Graph of processor speeds, 1945 to the present

Performance

NON–VON NEUMANN ARCHITECTURES (3 OF 6)

- **Non–Von Neumann architectures**
  - Other ways to organize computers
  - Most are experimental/theoretical, EXCEPT parallel processing
- **Parallel processing**
  - Many processing units operating at the same time
  - Supercomputers (in the past)
  - Desktop multicore machines and “the cloud” (in the present)
  - **Quantum computing** (in the future)
    - https://www.scientificamerican.com/article/how-close-are-we-really-to-building-a-quantum-computer/

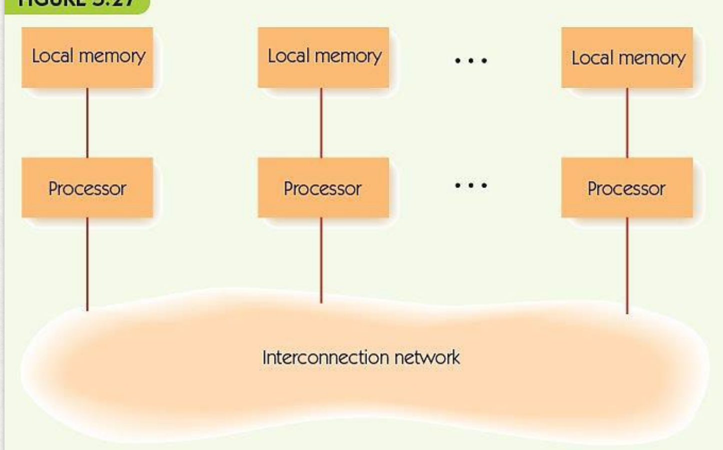
NON–VON NEUMANN ARCHITECTURES (4 OF 6)

MIMD parallel processing

- Multiple instruction stream/Multiple data streams
  - one approach cluster computing https://en.wikipedia.org/wiki/Computer_cluster
- Multiple, independent processors
- Each ALU operates on its own data
- Each processor can operate independently
  - On its own data
  - On its own program
  - At its own rate

NON–VON NEUMANN ARCHITECTURES (5 OF 6)

FIGURE 5.27



Model of MIMD parallel processing

NON–VON NEUMANN ARCHITECTURES (6 OF 6)

Varieties of MIMD systems

- Special-purpose systems: newer supercomputers
- **Cluster computing**: standard machines communicating over LAN or WAN
- **Grid computing**: machines of varying power, over large distances/Internet
  - Examples
    - SETI project - https://setiathome.berkeley.edu
    - BOINC at Berkley - https://boinc.berkeley.edu
- still research area ▶ **parallel algorithms**
  - Need to take advantage of all this processing power

SUMMARY (1 OF 2)

- Von Neumann architecture is standard for modern computing.
- Von Neumann machines have memory, I/O, ALU, and control unit; programs are stored in memory; execution is sequential unless program says otherwise.
- Memory is organized into addressable cells; data is fetched and stored based on MAR and MDR; uses decoder and fetch/store controller.

SUMMARY (2 OF 2)

- Mass data storage is nonvolatile; disks store and fetch sectors of data stored in tracks.
- I/O is slow, needs dedicated controller to free CPU.
- ALU performs computations, moving data to/from dedicated registers.
- Control unit fetches, decodes, and executes instructions; instructions are written in machine language.
- Parallel processing architectures can perform multiple instructions at one time.

AN INTRODUCTION TO SYSTEM SOFTWARE AND VIRTUAL MACHINES

CHAPTER 6

INTRODUCTION

- A **naked machine** has no tools or programs to help the user:
  - Write instructions in binary
  - Write data in binary
  - Load instructions into memory one cell at a time
  - Initiate running the program
- Quickly became too difficult for humans to do
- An interface had to be developed to hide the details and make the computer easier to control

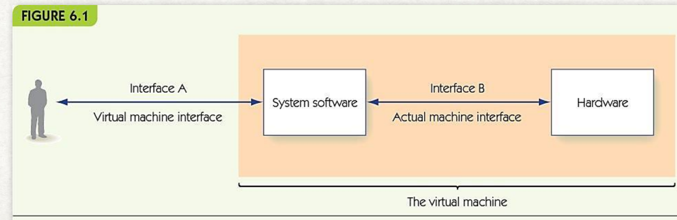
WOW!

Naked machine

1. Write program in binary
2. Load instructions one by one into memory
3. Insert start into memory address 0 and push “go” button
4. Read results from memory one by one, in binary

SYSTEM SOFTWARE

- **System software** is a collection of programs to:
  - Manage resources of the computer
  - Serve as intermediary between user and hardware
- System software creates a **virtual machine*** (or a **virtual environment**) that the user sees (abstraction)



The role of system software

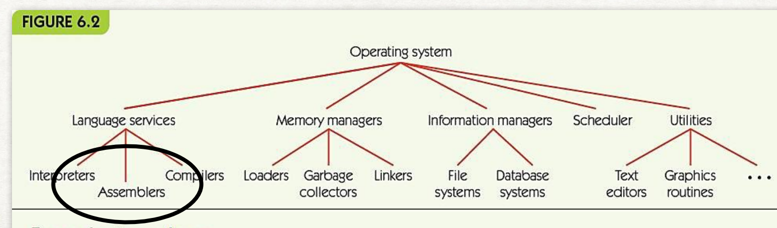
* not to be confused with virtual machines

OPERATING SYSTEMS

Operating system (covered in the second half of the course)

- single most important piece of software on the computer made up of:
  - The user interface
  - Memory managers
  - I/O systems
  - File and database systems
  - Utilities
  - Language services
  - Scheduler - process management
- Communicates with users, determines what they want, and activates other system programs, applications, packages, or user programs

LOTS OF SOFTWARE



Types of system software

- Language services support high level languages
- Memory managers allocate memory to programs
- Information managers organize mass storage
- Scheduler manages programs waiting to run
- Utilities: tools, including **program libraries**

WHEN WE PROGRAM

Virtual machine

1. Write program using text editor in high-level language
2. Save program to folder
3. Use translator to convert to binary
4. Use scheduler to load and run
5. Use I/O system to print results

REMEMBER MACHINE CODE → ASSEMBLY CODE

Low-level programming language is also called assembly language:

- Instructions map one-to-one to machine language
- Symbolic op codes (not binary)
- Symbolic addresses for instructions and data
- Pseudo-ops for data generation and more (data in human-friendly terms)
- Advantages over machine code
  - Clarity, readability, and maintainability
  - Can be placed at different locations in memory - position independent code

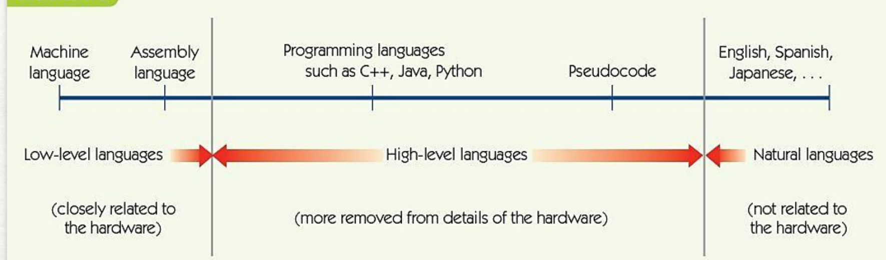
HIGH LEVEL LANGUAGES

High-level programming languages:

- Java, C++, and Python
- Advantages over assembly language:
  - More powerful
    - One high-level instruction may provide multiple machine instructions
  - User oriented
  - Not machine specific
  - Use both natural language and mathematical notation
  - Designed to help structure our programs

THE LANGUAGE CONTINUUM

FIGURE 6.3



The continuum of programming languages

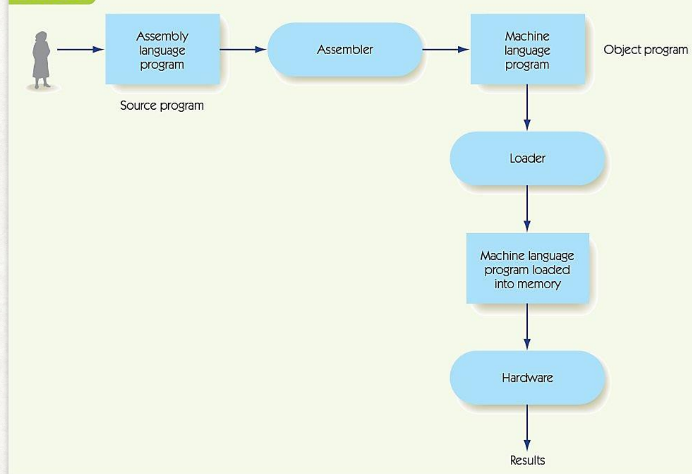
PROGRAMMING IN ASSEMBLY LANGUAGE

Assembly language process:

- **Source program (**assembly language)
- Translated by the **assembler** to
- **Object program** (machine language)
- Loader places in memory
- Hardware runs
- Results

THE PROCESS

FIGURE 6.4



The translation/loading/execution process

EXAMPLE INSTRUCTION

Example assembly language:

NEXTSTEP:	LOAD	X	-- Put X into reg. R

label: opcode mnemonic address field -- comment

- Label is optional name for this instruction's location
- Op code mnemonic and address field translate to machine language
- Comments are ignored by assembler—just for human use

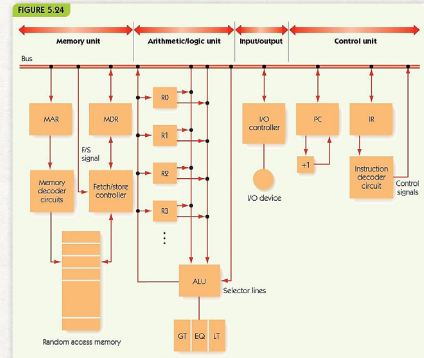
OUR ASSEMBLY LANGUAGE -AND OUR MACHINE*

FIGURE 6.5

Binary Op Code	Operation	Meaning
0000	LOAD X	CON(X) → R
0001	STORE X	R → CON(X)
0010	CLEAR X	0 → CON(X)
0011	ADD X	R + CON(X) → R
0100	INCREMENT X	CON(X) + 1 → CON(X)
0101	SUBTRACT X	R – CON(X) → R
0110	DECREMENT X	CON(X) – 1 → CON(X)
0111	COMPARE X	if CON(X) > R then GT = 1 else 0 if CON(X) = R then EQ = 1 else 0 if CON(X) < R then LT = 1 else 0
1000	JUMP X	Get the next instruction from memory location X.
1001	JUMPGT X	Get the next instruction from memory location X if GT = 1.
1010	JUMPEQ X	Get the next instruction from memory location X if EQ = 1.
1011	JUMPLT X	Get the next instruction from memory location X if LT = 1.
1100	JUMPNEQ X	Get the next instruction from memory location X if EQ = 0.
1101	IN X	Input an integer value from the standard input device and store into memory cell X.
1110	OUT X	Output, in decimal notation, the value stored in memory cell X.
1111	HALT	Stop program execution.

Typical assembly language instruction set

FIGURE 5.24



The organization of a Von Neumann computer

* Except our machine only has one Rregister